daVinci: Solarems Data Extraction

Christian Paulino

Teodor Talov

Instructor:

Dr. Januz Zalewski

CEN 4935

Software Project in Computer Networks

Florida Gulf Coast University

10501 FGCU Blvd. S.

Fort Myers, FL 33965-6565

Fall 2012

Draft #6

Submission Date: November 29, 2012

1. Introduction

This project, dubbed “daVinci”, is about creating a server that is dedicated to downloading data from `solarems.net` which is a FGCU solar plant website. The solar plant’s data are presented via CSV files (CSV stands for Comma-Separated Values). This type of file stores data in the form of text in a tabular format. Records are separated by line breaks and fields are separated by a comma. The physical server for the CSV files to be stored is an eBox-4864 Embedded PC.[5]

The eBox-4864 is a compact embedded PC compatible with Linux operating system. Ubuntu 10.10 was the operating system chosen for this project. The eBox-4864 runs on a Via Esther 1.2 GHz processor with 1 GB of RAM. It can connect to networks via an Ethernet connection and can also interface with other devices through 6 USB 2.0 ports. There are PS2 ports for mouse and keyboard connectivity, a VGA port for graphical interfaces, compact flash port, network interface card, and wireless adapter. All of these components are illustrated in Figure 1.

Fig.1. Back panel of eBox-4864.

(Source: http://test.watercolorgallery.eu/index.php?id=serveur)

There are several reasons this type of project may be useful. First of all technology is not always reliable, so it is good to have redundant backups of critical data. The project demonstrates a way to keep a backup of all important data from a website. In this case, it happens to be data on a solar plant from `solarems.net`. If that website were to be hacked or have its server crash, the data may be erased or inaccessible when it is needed. If there is a backup system set in place, data dependent operations may still continue and the website can have its data restored.

However, the main reason to retrieve the data from a remote solar plant server is to do analysis for potential faults of the plant equipment. Without constant monitoring of the equipment, a fault could go unnoticed and cripple the operation of a solar plant. It is important to be able to catch potential faults as soon as possible so the solar plant can continue to run smoothly.

2. Previous Accomplishments

This project is a continuation of one by C. Steiner [1]. During that project a client machine named daVinci was created with an objective to connect to the solar plant server and collect data. The eBox-4864 was setup and Ubuntu 10.10 was installed. It was connected to the FGCU network and all necessary interfaces were established. Such interfaces include the graphical interface on an external monitor and I/O devices. A shared folder was created on daVinci to store the CSV files. A screenshot of the page on the solarems.net website where the CSV files are downloaded from is shown in Figure 2.

In order to retrieve the CSV files, a Java program was created. This program uses a framework called HTMLUnit. It simulates a browser, in this case Firefox, and navigates through the `solarems.net` website to access the CSV files. The program was installed on the eBox-4864 to automate retrieving and storing the CSV files. Once installed, the Java program only needs to be executed once. It runs every hour and downloads all new CSV files since the last time it ran.

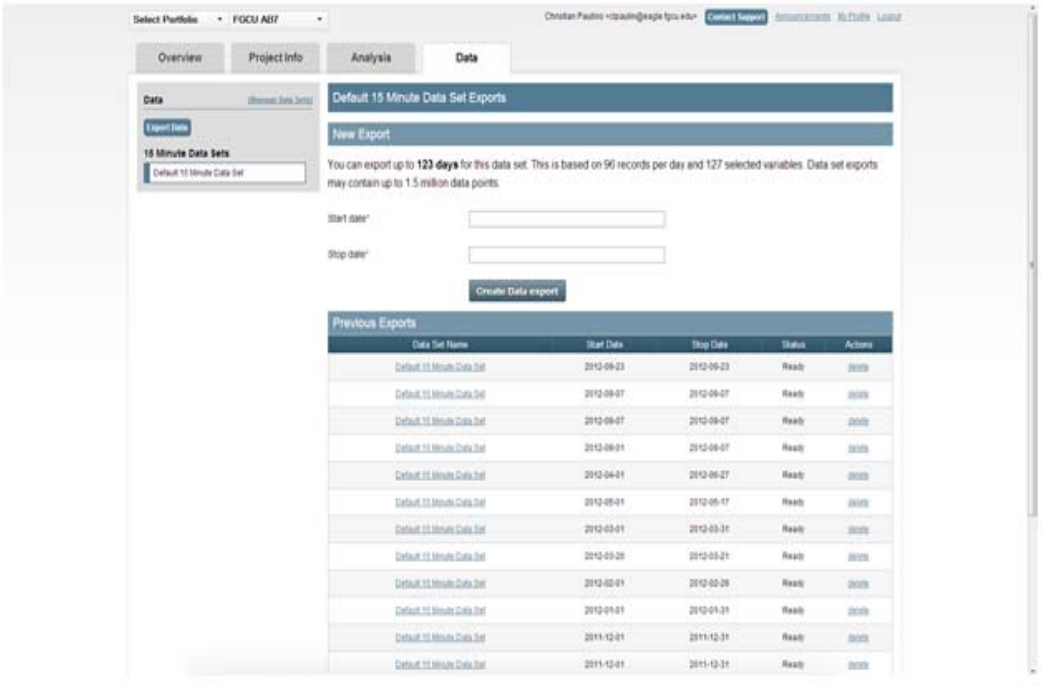


Fig.2. Sample Solarems.net data page.

3. Problem Description

The current implementation of daVinci no longer works. The Java program written to retrieve the CSV files can no longer do so. The `solarems.net` website changed the way it links to each CSV file. Errors of attempts to run the program are shown in Figure 3. The Java program that uses HTMLUnit does not compensate for the new signature added to each CSV file link. Besides no longer working, it wasn't very efficient to begin with. HTMLUnit does not support CSS or JavaScript, so it would produce errors when retrieving the CSV files. All daVinci did before it stopped working was store the CSV files. It did not provide a way to use the data in a useful way.

Because of these shortcomings, daVinci has to be re-implemented. A different technology has been chosen to perform the CSV file retrieval process. In order for the data to be used practically and not just stored, a database has to be used. The database will allow for the data to be manipulated and used beyond what a physical disk can provide. Drakerlabs, who host `slarems.net`, do not provide an API, which causes implementation to be more difficult.

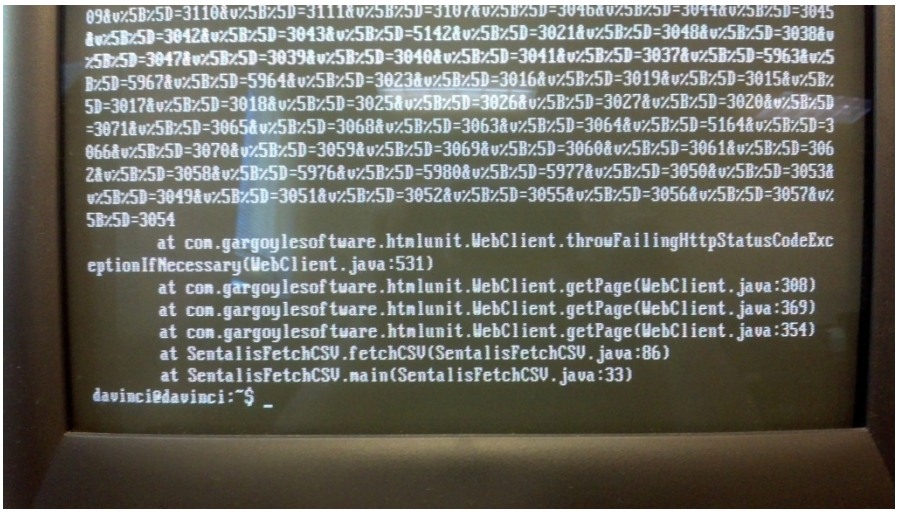


Fig.3. Old daVinci errors.

In summary, the following steps are proposed for new implementations:

- Use new technologies: Linux, Apache, MySQL, and PHP (LAMP)
- Store CSV files on eBox-4864
- Add CSV file data to a database

4. Solution and implementation

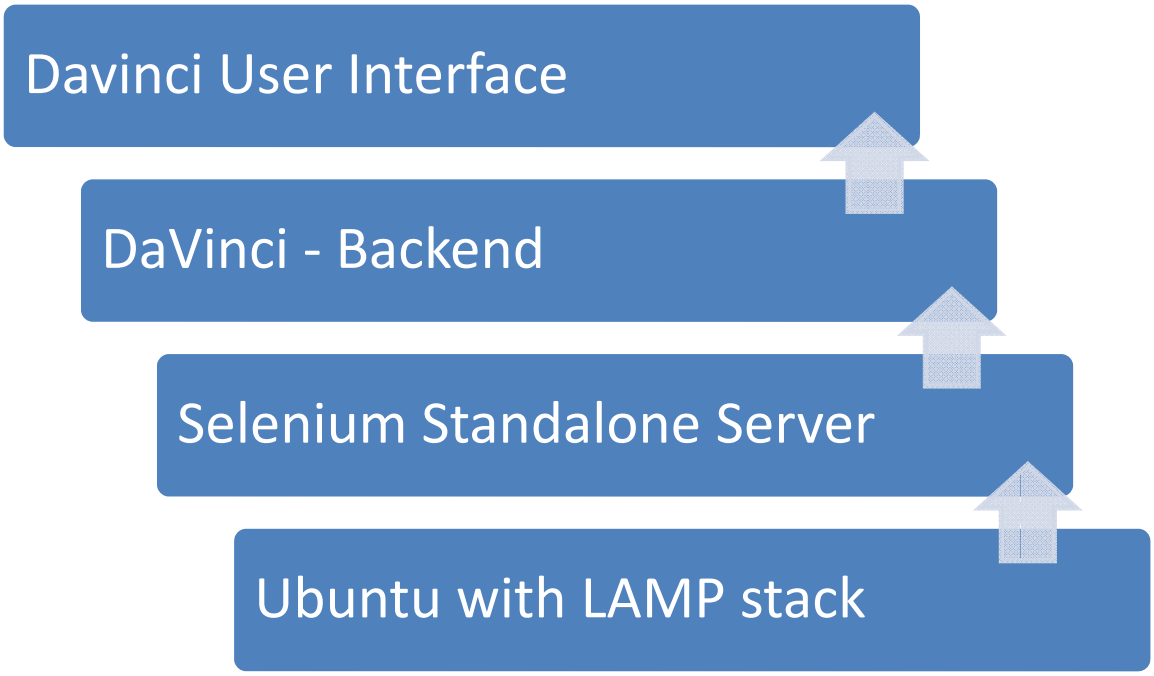


Fig 4. Technology Stack

Due to incompleteness prior implementation, it was decided that DaVinci will be re-implemented. Respective web technologies were chosen in order to provide internet based interface for users to interact with the system. Linux, Apache, MySQL, and PHP (LAMP) was selected as implementation stack (Fig 4).

Additionally, CakePHP [2] will be used as implementation framework due to its Rapid Application Development (RAD) capabilities. Specifically, CakePHP [2] provides out-of-the-box scaffolding features, which allow building basic User Interface (UI) so users can interact with data that are recorded by using CakePHP's [2] built-in Create, Read, Update, Delete (CRUD) methods. Additionally, CakePHP [2] provides Model-View-Controller (MVC) architecture which allows the project to be easily extended in the future. (Fig 4.) MVC architecture also provides clear separation of concerns; business logic is separate from data and presentation. MySQL database is used to store all data collected from SolarEMS.net service (Fig. 5). However, CakePHP [2] is database independent, so this does not impose limitation on how the data are stored or read. The database layer can be changed at any time without affecting the integrity of DaVinci.

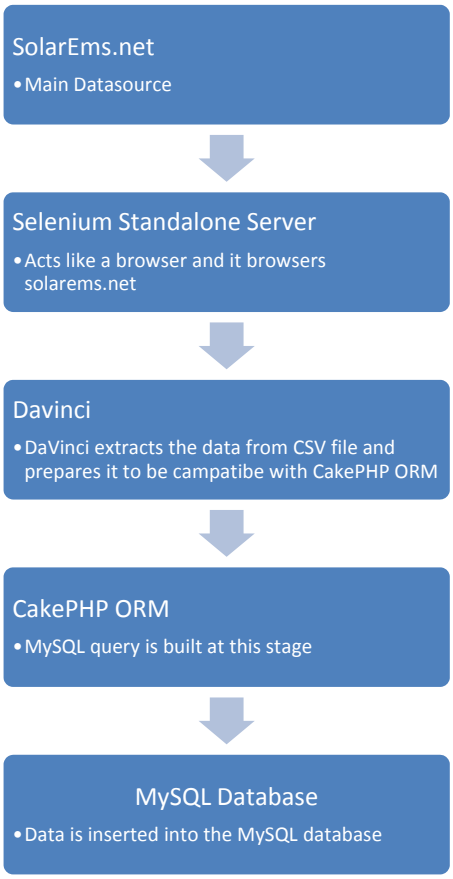
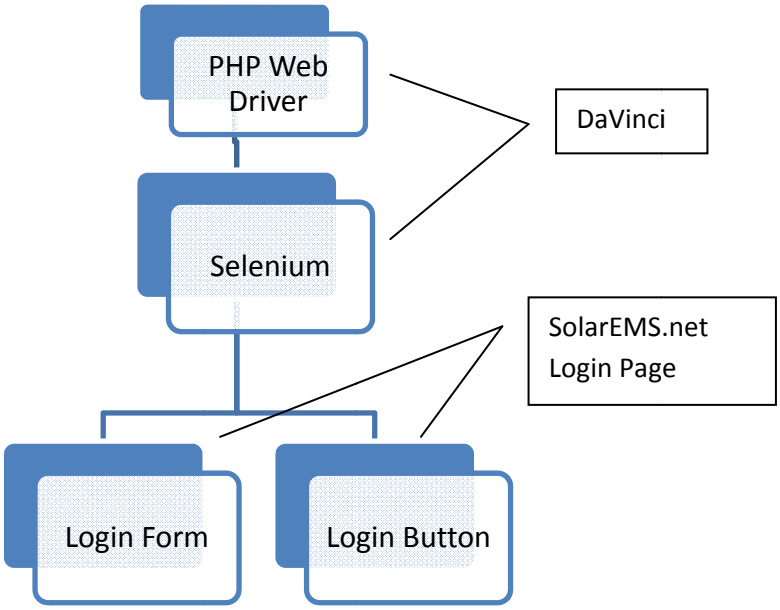

SolarEms.net
•Main Datasource
Selenium Standalone Server
•Acts like a browser and it browsers solarems.net
Davinci
•DaVinci extracts the data from CSV file and prepares it to be campatibe with CakePHP ORM
CakePHP ORM
•MySQL query is built at this stage
MySQL Database
•Data is inserted into the MySQL database

Fig 5. Dataflow diagram.

For data gathering Selenium standalone server [3] is used which provides browser-like capabilities. This allows DaVinci to browse SolarEMS.net as a regular user would, which will mitigate the risk of triggering any defence capabilities SolarEMS.net might have (Fig 6).



However, Selenium does not provide PHP interface, which means that PHP cannot work directly with it. Thus, a mediator is needed that will be able to provide the needed API to Selenium. Facebook's PHPWebDriver was chosen as a mediator.

PHPWebDriver provides basic API for interacting with Selenium, but it does provide enough support for this project. It supports basic events, such as JavaScript's getElementById(), which allows DaVinci to search SolarEMS's Document Object Model (DOM) and determine its state. This enables DaVinci to login and browse the website (Fig 7).

DaVinci will take advantage of the `cron` tables provided by Linux to execute itself every hour and update the database with the latest data available. This is done by taking advantage of CakePHP's [2] shell scripting capabilities.

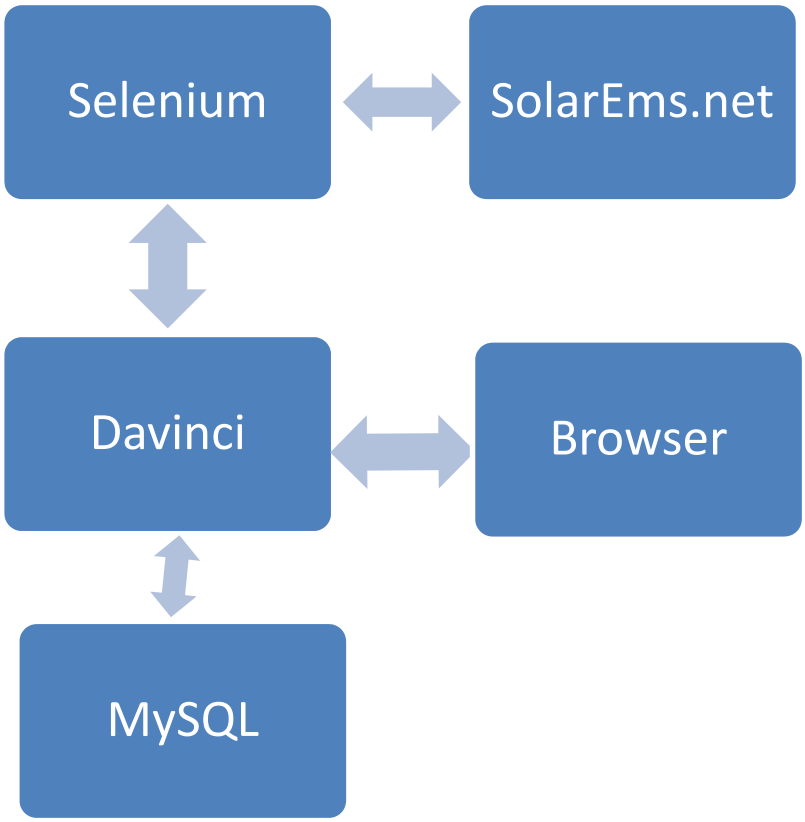


Fig 7. Physical Diagram

DaVinci development follows incremental development process (Fig 8).

- Installing prerequisites:
  - Ubuntu
  - PHP
  - MySQL
  - Apache
  - phpUnit
  - Selenium
- Developing modules:
  - DaVinci API to PHP Web Driver
  - DaVinci shell script

- DaVinci database layer
- DaVinci scaffolding

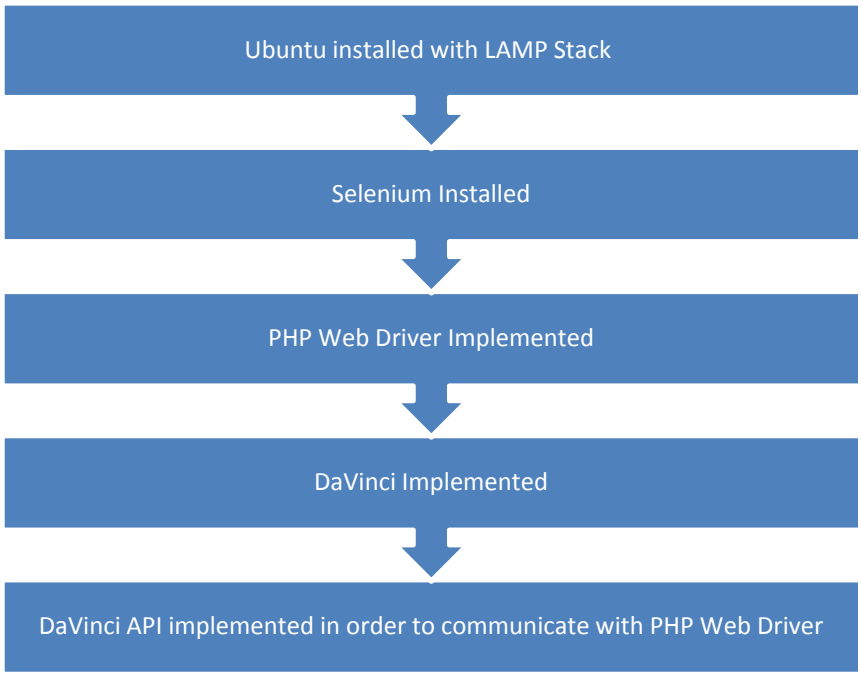

Ubuntu installed with LAMP Stack
Selenium Installed
PHP Web Driver Implemented
DaVinci Implemented
DaVinci API implemented in order to communicate with PHP Web Driver

Fig 8. Program Development Steps.

Testing of DaVinci is done via PHPUnitTest Framework [4]. This allows quickly pin-pointing possible technical issues as well as asserting that DaVinci is operational. In order to run the DaVinci test suite the following command is used in a terminal window (Fig 9):

```
php lib/Cake/Console/cake.php testsuite –app all
```

```
teodor@teodor-VirtualBox:/var/www/share/solar$ php lib/Cake/Console/cake.php testsuite app

Welcome to CakePHP v2.2.2 Console
---------------------------------------------------------------
App : app
Path: /var/www/share/solar/app/
---------------------------------------------------------------
CakePHP Test Shell
---------------------------------------------------------------
App Test Cases:
[1] Controller/ApiController
[2] Controller/DatasetsController
[3] Model/Dataset
What test case would you like to run?
[q] >
```

5. Experiments

First implementation was done using cUrl library (Fig. 9) to send a POST request to SolarEMS.net with credentials in order to attempt remote login. However, this approach proved to be unsuccessful and oversimplified, because SolarEMS.net did not respond as expected. Specifically, SolarEMS.net did not allow remote host to execute POST request and gain access to the system (Appendix A).

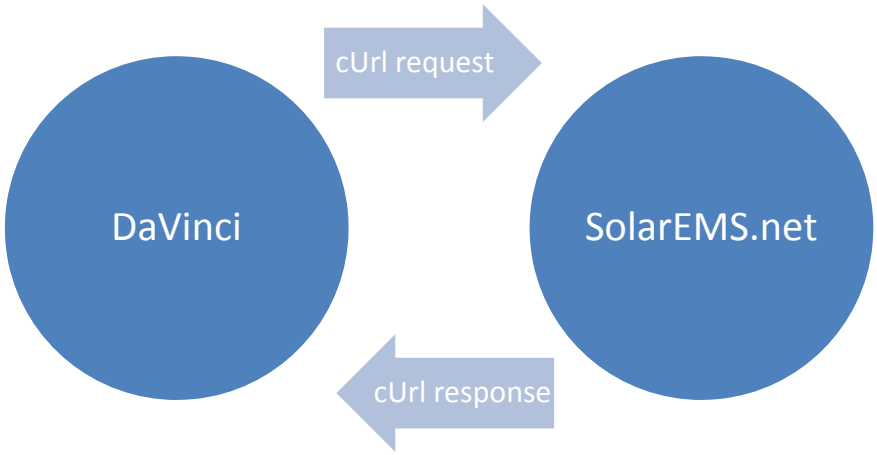


Fig 10. cURL request-response cycle.

A second implementation was attempted with using Selenium server and PHPWebDriver. This attempt was successful and proof of concept was achieved (Fig 10.) by developing a basic algorithm that logs in, downloads the latest CSV file available onto the local server. This affirms that the approach taken will work. Also, CakePHP's scaffolding options are enabled and a user is able to browse through the database.

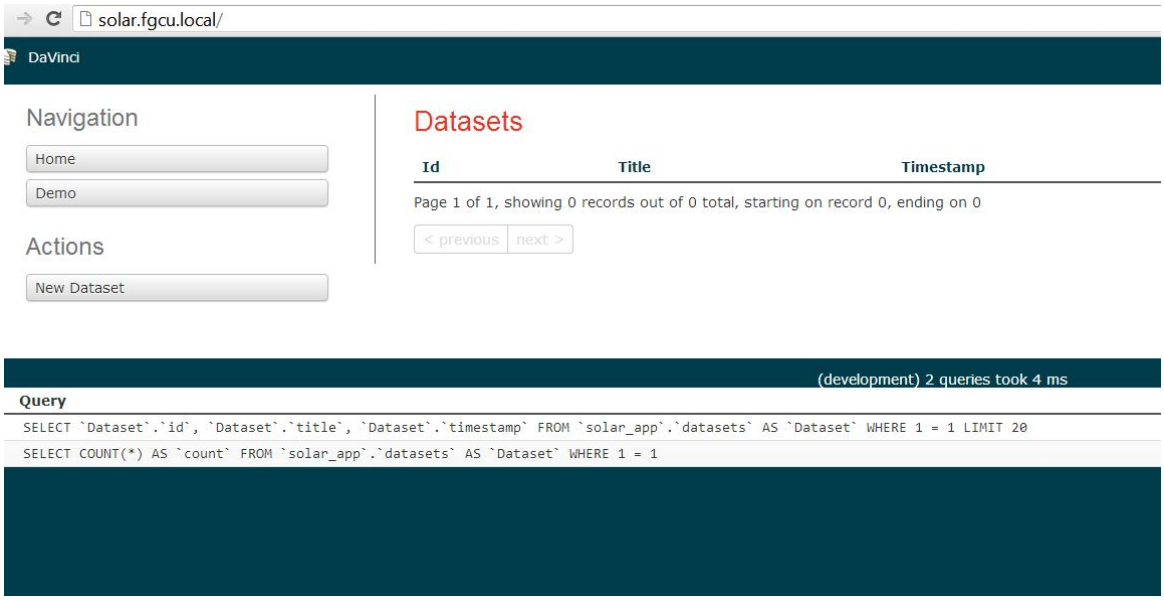


Fig 11. Snap shot of DaVinci, experimental stage.

Also, basic unit tests were implemented which provide the following assertions:

1. Selenium is up and running. This is done by calling the following URL via cURL:

   http://localhost:4444/selenium-server/driver/?cmd=testComplete

The expected response from Selenium is “OK”. If any other response is received the test will fail.

2. DaVinci can login to SolarEMS.net, credentials are valid as well as no change as been made to the DOM (Document Object Model) of SolarEMS.net. This is done after asserting that Selenium is working (Unit test 1 above). It is achieved by matching previously saved login form signature to the one that Seleium obtains on run time. If they match, there has been no change to the Solar EMS’s DOM and DaVinci can login.[screenshots will be included in future draft]

A database layer has been added with one table, which completes the Proof of Concept by combining business logic (obtaining the latest CSV file), database layer, and presentation.

Solar EMS's user interface was changed, which lead to the re-implementation of major part of DaVinci's shell scripts that are used to communicate with Selenium.

6. Conclusion

The old daVinci program quit working so a new daVinci program was written. The new program was written in php using the cake php framework. Php-webdriver from facebook and selenium were also used. The new program does more than just create a server for storing redundant data. It now puts all data into a database which can be accessed from a web page. This allows for easy access to the data as well as an easy way to get valuable information out of it. The program does this by navigating to the data page on solarems.net and opening the latest CSV file. The CSV file contains all the data produced from the solar plant. When it comes to a solar plant, data can be critical. Storing the data in a way that it can be accessed with ease will allow faults to be detected as well as other problems that the data may show. With a dedicated server automating the process of data retrieval, a lot of potential problems can be dealt with swiftly.

User Manual

The following steps must be taken in order to install and run the software.

1. Install any Linux distribution (Ubuntu preferred)
2. Install Apache 2.2.x
3. Install PHP 5.3 or greater
4. Install MySQL 5.5 or greater
5. Install Java
6. Enable support for PHP and MySQL in Apache
7. Install GIT
8. Install Firefox
9. Navigate to the following directory:

```
/var/www
```

10. Clone the following public repository:

    ```
    git@bitbucket.org:solarfgcu/solar2.git
    ```

    by using the following command:

    ```
    git clone git@bitbucket.org:solarfgcu/solar2.git .
    ```

    BitBucket account with public key required. Please create your account at www.bitbucket.org and add a public key to your account in order to authenticate.

11. Create a database named:

    ```
    solar_app
    ```

    By executing the following command:

    ```
    database create solar_app
    ```

12. Install application specific database schema by running the following command:

    ```
    php lib/Cake/Console/cake.php schema create
    ```

    Answer the prompt with "Y"

13. Start Selenium (in the backgorun) by running the following command, first navigate to /var/www and then run:

    ```
    java -jar selenium/selenium-server-standalone-2.25.0.jar &
    ```

14. Run the following command in order to start importing data from the solar plant:

    ```
    php lib/Cake/Console/cake.php import
    ```

References

[1] C. Steiner, " daVinci: eBox 4864 – Sentalis Fetch CSV Server," 2011.

[2] Cake Software Foundation, "CakePHP: the rapid development php framework. Pages," Cake Software Foundation, [Online]. Available: http://cakephp.org/. [Accessed 08 10 2012].

[3] "Selenium - Web Browser Automation," Selenium , [Online]. Available: http://www.seleniumhq.org. [Accessed 08 10 2012].

[4] S. Bergmann, "The PHP Unit Testing framework," [Online]. Available: https://github.com/sebastianbergmann/phpunit/. [Accessed 03 10 2012].

[5] Zentyal, Inc., "Zentyal - The Linux Small Business Server," Zentyal, Inc., [Online]. Available: www.zentyal.org. [Accessed 08 10 2012].

Appendix A

```
<?php
App::import('Vendor', 'php-webdriver/__init__');


class ImportShell extends AppShell {


    public $uses = array(
      'Dataset'
    );

    public $fields;
    public $solarData;

    public function main() {

      $this->out('Import Started.');
      $this->_import();
      $this->out('Import Complete.');
    }

    public function _import(){
    set_time_limit(0);

    $startDate = date('Y-m-d', strtotime('now'));
    $endDate = date('Y-m-d', strtotime('+1 days'));

    $webdriver = new WebDriver();
//     $session = $webdriver->session('htmlunit', array('javascriptEnabled' => true,
'version' => '3.6'));
    $session = $webdriver->session('firefox', array());
    $session->open("https://solarems.net");

    $session->element('id', 'user_session_email')->value(array('value' =>
str_split("ttalov@eagle.fgcu.edu")));
    $session->element('id', 'user_session_password')->value(array('value' =>
str_split("solarfgcu")));
    $button = $session->element('id', 'new_user_session');
    $button->submit();
    $session->open('https://solarems.net/projects/36-fgcu-ab7/data_sets/26/exports');

    $session->open('https://solarems.net/projects/36-fgcu-ab7/data_sets/26/exports');
    $session->element('id', 'data_export_start_date')->value(array('value' =>
str_split($startDate)));
    $session->element('id', 'data_export_stop_date')->value(array('value' =>
str_split($endDate)));

    $button = $session->element('id', 'new_data_export');
    $button->submit();

    sleep(20);
```

```
  $source = $session->source();

  $dom = new DOMDocument;
  @$dom->loadHTML($source);
  $primaryContent = $dom->getElementById('primary-content-with-nav');
  foreach ($primaryContent->getElementsByTagName('a') as $node){
    $this->download($node->getAttribute("href"));
    break;
  }

  //closing browser
  $session->close();
  }

  public function download($url = null){

  $row = 1;
  if (($handle = fopen($url, "rb")) !== FALSE) {

      while (($data = fgetcsv($handle, 1024*8, ",")) !== FALSE) {
          Cache::clear();
          $num = count($data);

          if($row == 1){
            $this->setFields($data);
          }else{
            $this->setSolarData($data);
          }

          if($row > 1){

            $insert = array_combine($this->getFields(), $this->getSolarData());
            $exists = $this->Dataset->find('list', array('conditions' => $insert));

            if(!empty($exists)){
                continue;
            }
            $this->Dataset->create();
            $this->Dataset->save($insert, array('validate' => false));
          }

          $row++;
      }
  fclose($handle);
  }
 }

     /**
* @return the $fields
*/
 public function getFields() {
   return $this->fields;
 }
```

```
      /**
 * @param field_type $fields
 */
  public function setFields($fields) {

    foreach ($fields as $value) {
      $this->fields[] = Inflector::camelize(Inflector::slug($value));
    }

//    $this->fields = array_unique($this->fields);
  }

  public function mysqlFields(){

    $fields = $this->getFields();
    $fields = array_unique($fields);

    foreach ($fields as $value) {

      if($value == 'local_timestamp' || $value == 'utc_timestamp'){
       $this->out('ALTER TABLE  `datasets` ADD  `'.$value.'` DATETIME NOT NULL;');
        continue;
      }

      $this->out('ALTER TABLE  `datasets` ADD  `'.$value.'` DOUBLE NOT NULL;');
    }
  }
/**
 * @return the $solarData
 */
  public function getSolarData() {
    return $this->solarData;
  }

/**
 * @param field_type $solarData
 */
  public function setSolarData($solarData) {
    $this->solarData = $solarData;
  }


}
```